Cincom

AD/ADVANTAGE

MANTIS SUPRA SQL Programming
OS/390, VSE/ESA

P39-3105-00

AD/Advantage® MANTIS SUPRA SQL Programming OS/390 VSE/ESA

Publication Number P39-3105-00

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, OH 45246-3732
U.S.A.

PHONE: (513) 612-2300
FAX: (513) 612-2000
WORLD WIDE WEB: http://www.cincom.com

Attention:

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

Release information for this manual

AD/Advantage MANTIS SUPRA SQL Programming, OS/390, VSE/ESA, P39-3105-00, is dated October 30, 2001. This document supports Release 5.5.01 of MANTIS SUPRA SQL Support.

We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. Please take the survey provided with the online documentation at your convenience.

Cincom Technical Support for AD/Advantage

All customers	Web:	http://supportweb.cincom.com
U. S. A. customers	Phone:	1-800-727-3525
	FAX:	(513) 612-2000 Attn: AD/Advantage Support
	Mail:	Cincom Systems, Inc. Attn: AD/Advantage Support 55 Merchant Street Cincinnati, OH 45246-3732 U. S. A.
Customers outside U. S. A.	All:	Visit the support links at http://www.cincom.com to find contact information for your nearest Customer Service Center.

Contents

About this book

Using this document

MANTIS SQL Support is used to create MANTIS applications that access SUPRA with SQL. This manual explains how to code MANTIS SQL programs. Beginning with an overview, the manual discusses the rules for combining MANTIS and SQL, programming considerations and dynamic MANTIS SQL. Sample programs are provided.

This manual supplements *MANTIS Language, OS/390, VSE/ESA*, P39-5002, and Cincom's documentation on SUPRA. System maintenance considerations for MANTIS SQL Support are in *MANTIS Administration, OS/390, VSE/ESA*, P39-5005.

Document organization

The information in this manual is organized as follows:

Chapter 1—Overview of MANTIS SQL support
Provides an overview of MANTIS SQL support and this manual.

Chapter 2—Embedding SQL statements in MANTIS programs
Describes how to code host and indicator variables in SQL statements, convert data between MANTIS support and the SQL database, and specify SQL data types in host variables.

Chapter 3—Programming considerations
Provides limits and specific considerations for MANTIS SQL support.

Chapter 4—Dynamic SQL statements
Describes the differences between dynamic SQL statements in MANTIS SQL Support and SQL in COBOL.

Appendix A—Sample MANTIS SQL support programs
Contains sample MANTIS SQL support programs.

Appendix B—Features not supported
Lists features not supported by MANTIS SQL.

Appendix C—Comparing SQL in MANTIS SQL support to SQL in COBOL
Provides general considerations applied to SQL in MANTIS SQL Support as compared to SQL in COBOL.

Appendix D—SQL keywords
Lists the SQL keywords used by MANTIS SQL support.

Index

Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example

`Constant width type`	Represents screen images and segments of code.	`Screen Design Facility`
		`GET NAME LAST`
		`INSERT ADDRESS`

Yellow-highlighted, red code	Indicates an emphasized section of code.	`00010 ENTRY COMPOUND`
		`00020 .SHOW"WHAT IS THE CAPITAL AMOUNT?"`
		`00030 .OBTAIN INVESTMENT`
		`00040 EXIT`

Slashed b (ⱥ)	Indicates a space (blank). The example indicates that a password can have a trailing blank.	`WRITEPASSⱥ`

Convention	Description	Example
Brackets [ ]	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations.	
	A single item enclosed by brackets indicates that the item is optional and can be omitted. The example indicates that you can optionally enter a program name.	`COMPOSE [program-name]`
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected. The example indicates that you can optionally enter NEXT, PRIOR, FIRST, or LAST. (NEXT is underlined to indicate that it is the default.)	**NEXT** **PRIOR** **FIRST** **LAST**
Braces { }	Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select. The example indicates that you must enter FIRST, LAST, or a value for *begin*.	**FIRST** ***begin*** **LAST**

Convention	Description	Example
Underlining (In syntax)	Indicates the default value supplied when you omit a parameter. The example indicates that if you do not specify ON, OFF, or a row and column destination, the system defaults to ON.	**SCROLL** [**ON** | **OFF** | [***row***] [**,*col***]]
	Underlining also indicates an allowable abbreviation or the shortest truncation allowed. The example indicates that you can enter either PRO or PROTECTED.	PROTECTED
Ellipsis points...	Indicate that the preceding item can be repeated. The example indicates that you can enter (A), (A,B), (A,B,C), or some other argument in the same pattern.	(*argument*,...)
UPPERCASE	Indicates MANTIS reserved words. You must enter them exactly as they appear. The example indicates that you must enter CONVERSE exactly as it appears.	CONVERSE *name*
Italics	Indicate variables you replace with a value, a column name, a file name, and so on. The example indicates that you can supply a name for the program.	COMPOSE [*program-name*]
Punctuation marks	Indicate required syntax that you must code exactly as presented. ( ) parentheses . period , comma : colon ; semicolon ' single quotation mark " " double quotation marks	[**LET**]*v* [(*i*) | (*i*, *j*)] [**ROUNDED**(*n*)] = *e1* [,*e2*,*e3*...]

MANTIS documentation series

MANTIS is an application development system designed to increase productivity in all areas of application development, from initial design through production and maintenance. MANTIS is part of AD/Advantage, which offers additional tools for application development. Listed below are the manuals offered with MANTIS in the IBM® mainframe environment, organized by task. You may not have all the manuals listed here.

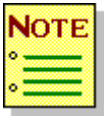
MASTER User tasks

- *MANTIS Installation, Startup, and Configuration, MVS/ESA, OS/390,* P39-5018
- *MANTIS Installation, Startup, and Configuration, VSE/ESA,* P39-5019
- *MANTIS Administration, OS/390, VSE/ESA*, P39-5005
- *MANTIS Messages and Codes, OS/390, VSE/ESA*, P39-5004*
- *MANTIS Administration Tutorial, OS/390, VSE/ESA,* P39-5027
- *MANTIS XREF Administration, OS/390, VSE/ESA*, P39-0012

General use

- *MANTIS Quick Reference, OS/390, VSE/ESA*, P39-5003
- *MANTIS Facilities, OS/390, VSE/ESA*, P39-5001
- *MANTIS Language, OS/390, VSE/ESA*, P39-5002
- *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013
- *MANTIS Messages and Codes, OS/390, VSE/ESA*, P39-5004*
- *AD/Advantage Programming*, P39-7001
- *MANTIS DB2 Programming, OS/390, VSE/ESA*, P39-5028

- *MANTIS SUPRA SQL Programming, OS/390, VSE/ESA*, P39-3105
- *MANTIS XREF, OS/390, VSE/ESA, OpenVMS*, P39-0011
- *MANTIS Entity Transformers*, P39-0013
- *MANTIS DL/I Programming, OS/390, VSE/ESA*, P39-5008
- *MANTIS SAP Facility, OS/390, VSE/ESA*, P39-7000
- *MANTIS WebSphere MQ Programming*, P39-1365
- *MANTIS Application Development Tutorial, OS/390, VSE/ESA*, P39-5026



Manuals marked with an asterisk (*) are listed twice because you use them for multiple tasks.

Educational material

AD/Advantage and MANTIS educational material is available from your regional Cincom education department.

1

Overview of MANTIS SQL support

MANTIS is an application development system for developing, testing, and executing applications interactively. MANTIS SQL Support is an extended version of MANTIS. It enables you to create MANTIS applications that access SUPRA by using SQL statements embedded in MANTIS programs. The presence of MANTIS SQL Support does not affect non-SQL MANTIS applications. MANTIS SQL Support programs can run side-by-side or with non-SQL MANTIS programs, with neither affecting the other.

Programming SQL in MANTIS SQL Support is similar to programming SQL in other programming languages. This manual refers to SQL in COBOL as representative of those languages. Because MANTIS is interpretive rather than compiled, some differences exist between MANTIS SQL Support and SQL in COBOL. These differences are noted in this manual and summarized in Appendix C.

Embedding SQL statements in MANTIS programs

You embed SQL statements in MANTIS programs as standard MANTIS comments. You must precede each SQL statement with an EXEC_SQL statement and follow it with an END statement, as shown below. Code the SQL statement text between these statements as MANTIS comments (each line must begin with a vertical bar (|), which is the MANTIS comment character).

An example of an SQL SELECT statement in a MANTIS program is shown below. Note that MANTIS automatically sets the indentation level (number of preceding periods) for all the statements.

```
04590 ..TEXT EMPL_NAME(30)
04600 ..BIG EMPL_NAME_IV
04610 ..EXEC_SQL
04620 ...| SELECT EMPLNAME
04640 ...| INTO :EMPL_NAME:EMPL_NAME_IV
04650 ...| FROM EMPLOYEE.TABLE
04660 ...| WHERE EMPLNAME = 'SMITH'
04670 ..END
04680 ..DO CLEAN_UP
```

MANTIS variables can appear in SQL statements (they are required by some SQL statements). MANTIS variables coded in SQL statements are called host variables. They transfer data between MANTIS and SUPRA. Host variables can be followed by indicator variables, which indicate the presence of NULL or truncated data. A colon (:) must precede all host and indicator variables coded in SQL statements. See “Embedding SQL statements in MANTIS programs” on page 17 for more information on host and indicator variables and embedding SQL statements in MANTIS programs.

Static and dynamic SQL statements

MANTIS SQL Support for SUPRA provides two basic types of SQL statements: static and dynamic.

Static SQL statements

Use static SQL statements when you know all the information required to execute the statement (SQL table name, column names, etc.) before executing the statement and the information will not change. SELECT, INSERT, UPDATE, and DELETE are examples of static SQL statements. In SQL in COBOL, you must code these statements in an application program, precompile and compile them before executing them.

Dynamic SQL statements

Dynamic SQL statements let you execute other SQL statements that have not been precompiled. Use dynamic SQL statements when you do not know all of the information needed to execute the SQL statement before executing the statement. An example is a query application that allows the user to input the SQL statement to be executed from an online terminal. PREPARE, DESCRIBE, EXECUTE, and EXECUTE IMMEDIATE are examples of dynamic SQL statements. In SQL in COBOL, you must precompile and compile dynamic SQL statements, but not the SQL statements they execute. Dynamic SQL statements provide flexibility, but require more computer resources and deliver less performance than static SQL statements.

Coding static and dynamic SQL statements

You can code both static and dynamic SQL statements in MANTIS programs. Both static and dynamic SQL statements can be contained in a single MANTIS program. For an explanation of how to code SQL statements in a MANTIS program, see “Embedding SQL statements in MANTIS programs” on page 17.

2

Embedding SQL statements in MANTIS programs

Rules for embedding SQL statements in a MANTIS program

You embed SQL statements in MANTIS programs within an EXEC_SQL-END block. Begin each SQL statement with EXEC_SQL and terminate with END. You must begin each line of SQL text between EXEC_SQL and END with the MANTIS comment character, the vertical bar (|). All SQL text within the EXEC_SQL-END block must conform to the rules of SQL syntax (rather than MANTIS syntax), except where host language syntax is permitted.

When you embed SQL statements in a MANTIS program, the following rules apply:

- Only one SQL statement can be present within an EXEC_SQL-END block.

```
..EXEC_SQL
...| OPEN C1
...| FETCH C1 INTO ...
...| CLOSE C1
..END
```

Invalid: Three SQL statements in the EXEC_SQL-END block.

- Any text between EXEC_SQL and END must be part of an SQL statement and must be preceded by a vertical bar (|). Once MANTIS SQL Support encounters a vertical bar, the rest of the program line is considered part of a single SQL statement. Other MANTIS statements or comments are not permitted.

```
..EXEC_SQL
...| OPEN C1
...OPENED=TRUE
..END
```

Invalid: A statement other than a comment is between EXEC_SQL and END.

```
..EXEC_SQL
...| OPEN C1:OPENED=TRUE
..END
```

Invalid: A MANTIS statement is appended to a valid SQL statement.

```
..EXEC_SQL
...| OPEN C1:|EMPLOYEE CURSOR
 ..END
```

Invalid: A comment is appended to a valid SQL statement.

- A colon within an EXEC_SQL-END block identifies a MANTIS host variable, not a new statement.

```
..BIG A
..EXEC_SQL
...| FETCH C1 INTO :A
..END
```

C1 is an SQL entity;

A is a MANTIS host variable.

- An SQL statement appended to an EXEC_SQL statement with a colon (the character that separates MANTIS statements) is part of the SQL statement; it is considered to be within the EXEC_SQL-END block.

```
..EXEC_SQL:| SELECT NAME
...| FROM EMPL_TABLE..
...| WHERE ZIP = '12345'...
..END
```

Valid

- In an SQL statement, multiple blanks at the beginning or end of an SQL statement are treated as a single blank.

```
..EXEC_SQL                    is equivalent to    ..EXEC_SQL
...|    OPEN C1                                  ...| OPEN C1
..END                                           ..END
```

All spaces between words on the same or different lines are compressed at every execution except those contained in a text literal.

```
..EXEC_SQL  equivalent to      ..EXEC_SQL
...|                           ...| SELECT COL1 FROM TABLE WHERE
...| SELECT                    ...| COL1=' ABCDEF '
...|  COL1                     ..END
...|  FROM TABLE
...|  WHERE
...|  COL1=' ABCDEF '
..END
```

Although multiple blanks can add to readability, they also incur additional processing overhead. You may want to avoid using them for this reason.

- An SQL statement in an EXEC_SQL-END block can be broken into multiple lines. MANTIS SQL Support reads the text on two consecutive comment lines in an EXEC_SQL-END block as if it were separated by a single blank (one SQL statement).

```
..EXEC_SQL                    is equivalent to    ...EXEC_SQL
...| OPEN                                        ...|OPEN C1
...| C1                                          ..END
..END
```

- Do not code SQL text literals on more than one line.

```
..EXEC_SQL
...| SELECT COL1
...| INTO :HOST_VAR
...| FROM TABLE
...| WHERE COL1='ABC
...| DEF'
..END
```

Invalid: An SQL text literal appears on more than one line.

- A MANTIS statement on the same line as the END statement in an EXEC_ SQL-END block is not executed. This is consistent with the rules for using END with MANTIS IF, WHILE, WHEN, and UNTIL statements. MANTIS comments are permitted.

```
..EXEC_SQL
...| OPEN C1
..END:OPENED=TRUE
```

OPENED=TRUE is disregarded.

```
..EXEC_SQL
...| OPEN C1
..END:| C1 IDENTIFIES TAG FILE ENTRIES
```

A valid comment.

Accessing multiple SUPRA databases From MANTIS

SUPRA permits up to eight databases to be accessed concurrently. In MANTIS SQL Support, you access multiple SUPRA databases by coding the number of the database on the EXEC_SQL statement, as shown below:

```
EXEC_SQL(n)
```

The database number (if present) must be a MANTIS expression that evaluates to an integer value between 1–8, inclusive. If the database number is not coded on the EXEC_SQL statement, it defaults to 1. When SQL text is appended to the EXEC_SQL statement, it must follow any SUPRA database number specified, as in the following example:

```
..EXEC_SQL(3):| SELECT .  .  .
..END
```

Coding host variables in SQL statements

Host variables are MANTIS data variables that are used to provide input or receive output from SUPRA. Host variables are identified in SQL statements by a colon (:) prefix. In the following example, EMPL is a host variable:

```
..SMALL EMPL
..EXEC_SQL
...| FETCH CURSOR1 INTO :EMPL
..END
```

Host variables can be explicitly declared as BIG, SMALL, TEXT, or KANJI. Like other MANTIS variables, undeclared host variables are implicitly declared when they are first used. Any previously undefined MANTIS variable referenced in an SQL statement is automatically declared as a MANTIS BIG variable.

```
..BIG A
..EXEC_SQL
...| FETCH C1 INTO :A
..END
```

is equivalent to

```
..EXEC_SQL
...| FETCH C1 INTO :A
..END
```

Host variables can either be input host variables or output host variables, depending on how they are used in the SQL statement. Input host variables contain data that SUPRA requires to perform the SQL statement. These are usually search condition values. EMPL2 is an input host variable in the example below. It contains the employee number of the specific employee to be selected.

Output host variables are variables that contain data requested from SUPRA. They are usually found in SELECT or FETCH statements. In the following example, EMPL1 is an output host variable since SUPRA will place the results of the SELECT statement there:

```
..TEXT EMPL1(30)
..SMALL EMPL2
..EXEC SQL
...| SELECT EMPLNO, EMPLNAME
...| INTO :EMPL1
...| FROM EMPL.TABLE
...| WHERE EMPLNO = :EMPL2
..END
```

Normally, host variables are MANTIS variables declared as BIG, SMALL, TEXT, or KANJI. Certain MANTIS functions and complex data types (such as SCREEN and FILE) can also be used as host variables, depending on how they are used in the SQL statements. The following table shows the MANTIS entities that can be used as SQL host variables.

Declared data variables	Valid for input host variable	Valid for output host variable
BIG	Yes	Yes
SMALL	Yes	Yes
TEXT	Yes[1]	Yes[1]
KANJI	Yes[1]	Yes[1]
Immediate numeric functions:		
DATAFREE	Yes	No
DOLEVEL	Yes	No
E	Yes	No
FALSE	Yes	No
PI	Yes	No
PROGFREE	Yes	No
TRUE	Yes	No
USERWORDS	Yes	No
ZERO	Yes	No

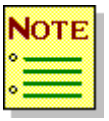
[1] Input host variables can include both array and substring subscripts. Output host variables can include array subscripts but cannot include substring subscripts.

Declared data variables	Valid for input host variable	Valid for output host variable
Immediate text functions:		
DATE	Yes	No
KEY	Yes	No
PASSWORD	Yes	no
PRINTER	Yes	No
TERMINAL	Yes	No
TERMSIZE	Yes	No
TIME	Yes	No
USER	Yes	No
Complex data types:		
ACCESS	Yes[2]	No
FILE	Yes[2]	No
INTERFACE	Yes[2]	No
SCREEN	Yes[2]	No
TOTAL	Yes[2]	No
VIEW	Yes[2]	No
Other		
ENTRY	No	No
PROGRAM	No	No

[2] These are treated as TEXT functions (status of last use).

A host variable can be located in a MANTIS array. You can use arithmetic expressions and MANTIS functions to specify subscripts of host variables. MANTIS syntax rules apply to subscripting, even though the subscript is coded in an SQL statement. In the example below, all text following the colon must conform to MANTIS syntax rules. For example:

```
..SMALL N,T
..BIG EMPL(20,40)
..EXEC_SQL
...| FETCH C1 INTO :EMPL(1+N,INT(T))
..END
```



Only the host variable, not other MANTIS variables referred to in subscript expressions, is prefixed with a colon. In the example above, the variables N and T are not prefixed with a colon, but are MANTIS variables used to calculate the subscript for the EMPL array.

Accessing host variables directly from within arrays is an extension to SQL made for MANTIS SQL Support. It may not be available in SQL in COBOL.

You can use host variables in SQL expressions. A colon (:) must precede each host variable, as shown in the following example:

```
..EXEC_SQL
...| INSERT INTO OWNER.TAB (COL-A)
...| VALUES (:SALARY * 1.1)
..END
```

Coding indicator variables in SQL statements

Indicator variables are host variables containing information about data being sent to or received from SUPRA. Indicator variables indicate whether data transferred between MANTIS and SUPRA was transferred successfully, is NULL, or was truncated. If the data was transferred successfully, the indicator variable is zero. If the data was NULL, the indicator variable contains a negative value. If the data was truncated, the indicator variable contains the untruncated length of the data.

Indicator variables are optional, but if used, they must be prefixed with a colon and immediately follow the corresponding host variable (or subscript expression). In the following example, EMPLIV and NAMEIV are indicator variables:

```
..EXEC_SQL:| SELECT EMPLNO, EMPLNA
...| INTO :EMPL(15,3):EMPLIV, :NAME:NAMEIV
...| FROM EMPLOYEES WHERE DEPT = 17
..END
```

Like host variables, indicator variables can be defined explicitly or implicitly. Only SMALL and BIG variables can be used as indicator variables. If indicator variables are declared implicitly, they will default to MANTIS BIG.

You can also use an indicator variable to insert a NULL value into a SUPRA table column. Storing a negative value in an indicator variable causes SUPRA to make the value of the associated table column NULL, regardless of the contents of the host variable. In the following example, a NULL value is inserted for column COLA (regardless of the value of VAR) when the INSERT statement is executed:

```
..VARIV=(-1)
..EXEC_SQL
...| INSERT INTO OWNER.TAB (COLA,COLB)
...| VALUES (:VAR:VARIV, :XV:XYIV)
..END
```

Indicator variables are normally optional. However, they are required when an SQL table column is to be set to NULL, as shown above.

When accessing SUPRA, you may need to check the indicator variable before using the data returned by SUPRA. If a table column is NULL, the value of the associated host variable is not defined. The contents of the MANTIS host variable may be unchanged or may be cleared depending on options selected when MANTIS was installed. Check with your system administrator for more information.

Converting data between MANTIS SQL support and SUPRA

Within MANTIS SQL Support, data is always maintained in MANTIS data types (BIG, SMALL, TEXT, KANJI). If data conversion is required, it is performed by SUPRA. The following table lists permissible data type conversions. Any combination of MANTIS and SQL data types not listed in this table may result in run-time errors. Note that loss of precision, numeric overflow, and data truncation are possible during data conversion.

The following table shows valid data type conversions:

SQL data type	MANTIS data type	Notes
0 (Fixed)	BIG/SMALL	Loss of precision may occur when converting from SQL to MANTIS. Overflow may occur when converting from MANTIS to SQL.
1 (Float)	BIG/SMALL	Loss of precision may occur when converting from SQL to MANTIS. Overflow may occur when converting from MANTIS to SQL.
2 (Character)	TEXT	When converting in either direction, truncation may occur.
3 (Byte)	TEXT	When converting in either direction, truncation may occur.
4 (Date)	TEXT	When converting in either direction, truncation may occur.
5 (Time)	TEXT	When converting in either direction, truncation may occur.
6 (String)	TEXT	When converting in either direction, truncation may occur.
10 (DBYTE)	KANJI	When converting in either direction, truncation may occur.

3

Programming considerations

MANTIS SQL Support allows you to execute SQL statements from a MANTIS program. SQL statements are executed in a MANTIS program as standard MANTIS comments, enclosed in an EXEC_SQL-END block. As MANTIS encounters each SQL statement, it prepares it for execution and then executes it, in effect performing the same steps (preprocess, compile, link, and execute) that are performed by programs in SQL in COBOL that contain embedded SQL statements. However, unlike programs in SQL in COBOL, MANTIS programs can be modified (including the SQL statements) and then immediately re-executed.

Before you begin writing MANTIS SQL Support programs, review the programming considerations discussed in this chapter:

- The scope of an SQL cursor or statement is local to a program. Because both are SQL entities and not MANTIS entities, you cannot pass them as parameters or use them in non-SQL MANTIS statements (see “Scope of SQL cursors and statements” on page 29).
- The WHENEVER statement in MANTIS SQL Support differs slightly from the WHENEVER statement in other SQL in COBOL (see “SQL WHENEVER statement” on page 30).
- Elements in the SQLCA (SQL Communications Area) are accessed through a MANTIS function called SQLCA, rather than as elements of an SQLCA data structure as in SQL in COBOL (see “SQLCA in MANTIS SQL support” on page 35).
- MANTIS HPO (High Performance Option) binding is terminated by any SQL statement or function (EXEC_SQL, SQLCA, SQLDA) (see “Binding with the high-performance option (HPO)” on page 39).
- The effects of the SQL COMMIT WORK and ROLLBACK WORK statements differ slightly from the COMMIT and RESET statements in MANTIS (see “SUPRA's COMMIT and ROLLBACK versus MANTIS SQL support's COMMIT and RESET” on page 39).
- Running a program from a line number can have unpredictable results (see “Running a program from a line number” on page 40).
- Error messages can come from three different sources: MANTIS SQL Support, the MANTIS nucleus, and SUPRA (see “Error messages” on page 41).
- MANTIS SQL Support does not limit the number of host variables in an SQL statement. However, this number may be limited by SUPRA (see “Maximum number of host variables” on page 42).
- Truncation of TEXT data can occur when parameters of MANTIS ENTRY statements are used as host variables (see “Using ENTRY statement parameters as host variables” on page 42).
- Certain SQL keywords cannot be used as MANTIS host variable names (see “SQL keywords” on page 79).

Scope of SQL cursors and statements

The scope of SQL cursors and statements is local to the program and do-level where they are defined. SQL cursors and statements are global entities to SUPRA. However, attempting to use SQL cursors and statements outside of the program or do-level where they are defined in MANTIS SQL Support can cause problems.

MANTIS SQL Support maintains all SQL-related resources by the MANTIS program or do-level. A MANTIS program or do-level cannot access the SQL resources of another program or do-level. If SQL resources are defined in a program and used in another program or do-level, MANTIS SQL Support may not be able to access the SQL statement.

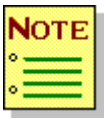
For example, if an SQL DECLARE statement is coded in a MANTIS program or do-level and the associated OPEN for the DECLARE is coded in another program or do-level, MANTIS SQL Support may not be able to execute the OPEN statement in all circumstances because the OPEN statement must have access to the DECLARE statement text. SUPRA can discard SQL statements based on certain time limits. If a discarded SQL statement is executed, SUPRA returns an error code indicating that the SQL statement is invalid and must be prepared again (parsed) for execution. If this occurs on the OPEN statement, there is no way to access the DECLARE statement text to prepare it again for execution.

SQL WHENEVER statement

The WHENEVER statement in MANTIS SQL Support differs from the WHENEVER statement in SQL in COBOL in four ways. In MANTIS SQL Support:

- The WHENEVER statement is interpretive, not compiled.
- GOTO is replaced by DO.
- STOP is replaced by FAULT.
- The default action for SQLERROR is FAULT, not CONTINUE.

The syntax of the MANTIS SQL Support WHENEVER statement is shown below. Note that any action (DO, FAULT, or CONTINUE) can be selected for any condition (SQLERROR, SQLEXCEPTION, SQLWARNING, NOT FOUND).



If you do not code the WHENEVER statement, the conditions will default to the actions specified in the considerations below. If you do code the WHENEVER statement, you must code both condition and action.

WHENEVER *condition action*

condition

Description *Required*. Indicates the condition you want to check for.

Options SQLERROR, SQLEXCEPTION, SQLWARNING, and NOT FOUND

SQLERROR

Description *Optional*. Specifies that SUPRA detected an error. The SQL statement execution was not successful, and the SQLCA SQLCODE contains a negative value.

Consideration If you do not specify a WHENEVER SQLERROR, the default action is FAULT.

SQLEXCEPTION

Description *Optional*. Indicates SQL timeout error conditions; SQLCODE>100. Timeouts can be configured for locks and access to SUPRA.

Consideration If you do not specify a WHENEVER SQLEXCEPTION, the default action is CONTINUE.

SQLWARNING

Description *Optional*. Specifies that SUPRA detected a condition that may require program intervention. The SQL statement execution was successful. The SQLCA SQLCODE may contain a positive value less than 100 and/or one or more of the SQLCA SQLWARN flags may contain nonblank characters.

Consideration If you do not specify a WHENEVER SQLWARNING, the default action is CONTINUE.

NOT FOUND

Description *Optional*. Specifies that SUPRA cannot find a row to satisfy your SQL statement, or there are no more rows to be retrieved. The SQLCA SQLCODE contains 100.

Consideration If you do not specify a WHENEVER NOT FOUND, the default action is CONTINUE.

action

Description *Required*. Specifies the action to be taken when the named condition is met.

Options DO entry-name[(parms)], FAULT, and CONTINUE.

DO

Description *Optional*. Specifies a standard MANTIS DO (internal or external) and corresponds to the WHENEVER-GOTO SQL statement in SQL in COBOL. WHENEVER-DO transfers control to the specified internal routine or external program whenever the named condition is encountered.

Considerations

- WHENEVER-DO can transfer control to an internal routine or external program, which in turn can contain any MANTIS logic, including CHAIN, EXIT, or STOP statements. The current values of any DO arguments are passed to the named subroutine or external program. The subroutine or external program EXIT returns control to the next statement following the EXEC_SQL that caused the DO to occur.
- The WHENEVER-DO action resembles the existing functionality of the TRAP statement in MANTIS. If the DO portion of a WHENEVER-DO contains an error, MANTIS returns a MANTIS error message associated with the DO statement, not an SQL WHENEVER-type error. MANTIS displays the line in error in the subroutine or external program. The WHENEVER statement may be outside of the current execution path. Remember that DO is executed as a result of an SQL statement detecting the condition with which the DO action is associated.

FAULT

Description *Optional*. Terminates execution of the program and displays the error message returned by SUPRA in the form of a MANTIS fault message. FAULT corresponds to the WHENEVER-STOP SQL statement in SQL in COBOL.

CONTINUE

Description *Optional*. Permits program execution to continue without interruption when the named condition occurs. Program execution continues with the statement following the EXEC_SQL statement in the MANTIS program. Your program should then check the SQLCA SQLCODE for the results of each SQL statement execution.

The following table shows the default action for each condition when the WHENEVER statement is not coded:

Condition	Default action
SQLERROR	FAULT
SQLEXCEPTION	CONTINUE
SQLWARNING	CONTINUE
NOT FOUND	CONTINUE

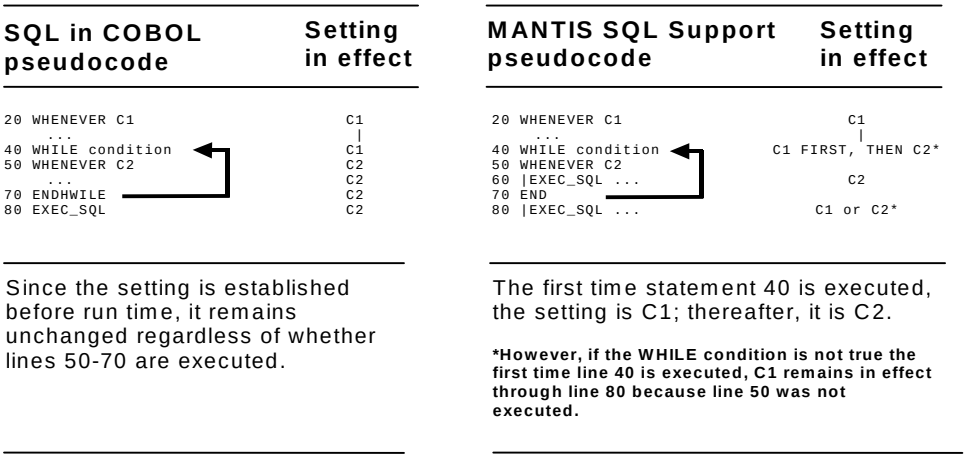
Example

```
00230 EXEC_SQL
00240 | WHENEVER SQLERROR
00250 | DO ERROR_ROUTINE(PARM1,PARM2,PARM3)
00260 END
00270 EXEC_SQL: | WHENEVER SQLEXCEPTION FAULT
00280 END
```

Using SQL WHENEVER as a declarative statement

In SQL in COBOL, the SQL WHENEVER statement is a declarative statement. It is processed when the program is precompiled, not when it is executed. Consequently, in SQL in COBOL the current SQL WHENEVER setting is determined by its sequential position in the program.

In MANTIS SQL Support, the WHENEVER statement is an executable statement. The last-executed WHENEVER statement is in effect regardless of its sequential position in the program. This difference is important when a WHENEVER statement is used with MANTIS conditional statements. The following figure illustrates the different effects of a declared versus executed WHENEVER statement. "C" denotes a condition and "1" and "2" denote actions. The same considerations apply to UNTIL, WHEN, and IF structures in MANTIS.



Scope of the WHENEVER statement

The scope (range) of the WHENEVER statement is every EXEC_SQL statement in the current MANTIS DO-level until a new WHENEVER statement is executed. To change the default WHENEVER settings, explicitly execute the WHENEVER statement in every external subprogram.

SQLCA in MANTIS SQL support

In SQL in COBOL, the SQLCA (SQL Communications Area) is a data structure. SQL in COBOL accesses elements in the SQLCA as items of data. In MANTIS SQL Support, these elements are accessed through the SQLCA function. The SQLCA function and statement perform the complementary operations of moving data to and from elements of the SQLCA structure. All standard SQLCA capabilities are provided as with SQL in COBOL.

The SQLCA statement stores data from the MANTIS program into the SQLCA. SQLCA is both a statement and a function. The SQLCA statement stores data from the MANTIS program into the SQL Communication Area (SQLCA). The SQLCA built-in function (read) transfers data from the SQLCA to the MANTIS program. For more detailed information, refer to *MANTIS Language, OS/390, VSE/ESA*, P39-5002.

The SQLCA statement is shown below followed by the SQLCA function. The SQLCA statement stores data from the MANTIS program into the SQL Communication Area (SQLCA).

SQLCA(*sqlca_element_name*)=*expression*

sqlca_element_name

Description *Required*. Specifies the element of the SQLCA that is to be transferred.

Format Must be a text literal or expression that evaluates to one of the SQLCA element names in the following tables.

expression

Description *Required*. Specifies the data to be transferred into the SQLCA.

Format Must be consistent with the datatype of the SQLCA element, text or numeric.

Consideration Certain SQLCA elements are read-only and cannot have data stored into them by the MANTIS program.

General considerations

- Some SQLCA elements are not present in the SQL SQLCA. These are extensions to the SQLCA unique to MANTIS. They are: DBTYPE and DBNAME. DBTYPE returns “SUPRA” to the MANTIS program. DBNAME is used to retrieve or set the SUPRA database value.

- Although data can be stored in some SQLCA elements, doing so does not pass any information to the SQL database. The SQLCA is returned to the MANTIS program after each SQL statement is executed (EXEC_SQL-END). Any data stored in the SQLCA will be overwritten when the next SQL statement is executed.

Example This example shows how a MANTIS program can retrieve SQL error message text for an SQLCA error message. The SQLCA statement is used to store the SQLCODE value in the SQLCA.

```
00150 TEXT SQL_ERROR_TEXT(254)
00160 SQLCA("SQLCODE")=-504
00170 SQL_ERROR_TEXT=SQLCA("SQLERRMC")
00180 END
```

The SQLCA built-in function, shown below, transfers data from the SQL Communication Area (SQLCA) into the MANTIS program.

SQLCA(*sqlca_element_name*)

sqlca_element_name

Description *Required*. Specifies the element of the SQLCA that is to be transferred.

Format Must be a text literal or expression that evaluates to one of the SQLCA element names in the following tables.

General considerations

- Some SQLCA elements are not present in the SQL SQLCA. These are extensions to the SQLCA unique to MANTIS. They are DBTYPE and DBNAME. DBTYPE returns “SUPRA” to the MANTIS program. DBNAME returns the name of the SUPRA database currently in use.
- If an SQLCA TEXT element is moved to a MANTIS variable of shorter length (for example, an 8-character SQLCA element to a 6-character MANTIS variable), the right-most characters are truncated.

Example This example shows how data is retrieved from the SQLCA by the SQLCA built-in function. Line 160 checks the SQLCA SQLCODE to determine if all table rows have been fetched.

```
00130 EXEC_SQL
00140 . | FETCH C1 INTO :EMPL_NAME, :EMPL_NUMBER
00150 END
00160 IF SQLCA("SQLCODE")=100
00170 .DO END_OF_DATA
00180 END
```

Element name	MANTIS compatible data type	Contents / considerations	Updateable?
SQLCAID	TEXT(8)	Eyecatcher. Set by SQL	No
SQLCABC	BIG	Length of SQLCA. Set by SQL.	No
SQLCODE	BIG	Code indicating results of SQL statement execution.	Yes
SQLERRML	BIG	Length of SQL error message text.	No
SQLERRMC	TEXT(70)	SQL error message text.	Yes
SQLERRP	TEXT(8)	SQL diagnostic data.	Yes
SQLERRDn	BIG	SQL diagnostic data. n ranges from 1–6.	Yes
SQLWARNn	TEXT(1)	SQL warning flags. n ranges from 0–F.	Yes
DBTYPE*	TEXT(6)	Returns SQL database in use.	No
DBNAME*	TEXT(64)	Returns or sets SUPRA database name.	Yes

* This element is a MANTIS extension to the SQLCA. It is not present in the SQL SQLCA.

Binding with the high-performance option (HPO)

You can HPO bind MANTIS SQL Support programs using the standard MANTIS BIND command. The binding process stops when EXEC_SQL, SQLCA, or SQLDA are encountered; they are not bindable statements. The SQLCA function is discussed in "SQLCA in MANTIS SQL support" on page 35; the SQLDA function in "Dynamic SQL statements" on page 45. Finding (part of the MANTIS High-Performance Option) is discussed in the *MANTIS Program Design and Editing, OS/390, VSE/ESA*, P39-5013.

SUPRA's COMMIT and ROLLBACK versus MANTIS SQL support's COMMIT and RESET

In MANTIS SQL Support, using an SQL COMMIT or ROLLBACK does not imply a MANTIS COMMIT or RESET, but using a MANTIS COMMIT or RESET does imply an SQL COMMIT or ROLLBACK. Executing an SQL COMMIT statement only affects SUPRA and not MANTIS resources. An SQL COMMIT or ROLLBACK only affects the SUPRA database that is currently accessed. A MANTIS COMMIT or RESET affects all SUPRA databases.

MANTIS COMMIT and RESET statements, whether implicit (at terminal I/O) or explicit (coded in the MANTIS program) only execute an SQL COMMIT or RESET for SUPRA databases that were updated during the current Logical Unit of Work (LUW). If no updates were done, no COMMIT or RESET is executed. This avoids unnecessary database activity and allows you to use the KEEP LOCK option of the SQL COMMIT WORK statement to maintain SUPRA locks from one LUW to another.

Because MANTIS SQL Support does not execute an SQL COMMIT, if no updates are done, SUPRA read locks may not be freed at terminal I/O. You may need to include a COMMIT statement in your program to free these locks.

For more information on the KEEP LOCK option of the SQL COMMIT WORK statement, refer to the *SUPRA DRDM Application Programmer's Guide*, P26-2455, and the *SUPRA DRDM SQL Commands Reference Manual*, P26-2420.

Running a program from a line number

In MANTIS SQL Support you can run programs from a line number. However, this action can produce unpredictable results when the program contains SQL statements. Some SQL statements (such as FETCH) require that other SQL statements (such as DECLARE and OPEN) be previously executed. Running a program from a line number can cause errors because SQL statements have not been executed in the proper sequence. Also, SQL entities (cursors) can be affected by the MANTIS program display when running in MANTIS programming mode (the screen display causes a terminal I/O COMMIT, which causes an SQL COMMIT WORK, which can cause all SQL cursors to be closed).

Error messages

When using MANTIS SQL Support, you can receive messages from three sources:

- MANTIS nucleus
- MANTIS SQL Support
- SUPRA

Messages from the MANTIS nucleus and SQL support are documented in *MANTIS Messages and Codes, OS/390, VSE/ESA*, P39-5004. Messages from SUPRA are documented in the appropriate SQL manuals.

MANTIS SQL Support messages have a seven-character code like other MANTIS error messages. All MANTIS SQL error codes contain the letter "Q" in the fourth position, as shown in the example below:

```
NUCQFKE: SPECIFIED SQL HOST VARIABLE ELEMENT IS UNINITIALIZED
```

The way MANTIS SQL Support messages are displayed depends on whether the error was detected in the MANTIS Full Screen Editor, the MANTIS Line Editor, or a MANTIS program.

- **In the MANTIS Full Screen Editor.** If the message is too long to be displayed in full on the Message Line at the top of the screen, the message is displayed in a separate window at the bottom of the screen.
- **In the MANTIS Line Editor.** The MANTIS statement where the error was encountered is displayed at the bottom of the edit screen, followed by the error code and message. If the message is too long to fit on one line, the remaining text is not displayed.
- **In a MANTIS program.** The message appears at the bottom of the screen. It wraps to succeeding lines if necessary. The number of the statement causing the error and the MANTIS program name appear beneath the error message.

Messages from SUPRA contain the 3-character code QDB. A message from SUPRA contains the SQLCA SQLCODE value and its associated text message. The format is:

```
NUCQDBE:±nnnn:message-text
```

where ±nnnn is the SQLCA SQLCODE value and message-text is the message returned from SUPRA. For example:

```
NUCQDBE:-3008:INVALID KEYWORD OR MISSING DELIMITER
```

is returned from SUPRA if an SQLCODE of -3008 is returned to the SQLCA and the WHENEVER SQLERROR condition was set to FAULT.

Maximum number of host variables

The number of host variables that can be defined in a MANTIS program is limited only by SUPRA. MANTIS SQL Support does not limit the number of host variables that can be defined in a MANTIS program, other than the 2048 limit of symbolic names in a single MANTIS program..

Using ENTRY statement parameters as host variables

If ENTRY statement parameters are used as host variables in SQL statements in a MANTIS program, be certain that the lengths of any TEXT parameters passed do not change from one invocation to the next.

MANTIS SQL Support will use and maintain the length received on the first execution. If a TEXT parameter of a different length is used on a subsequent execution, the new length will be ignored. If the new length is larger than the existing length, MANTIS will continue to return the existing length, in effect truncating the data.

The following example shows ENTRY parameters used as host variables. In this case, MANTIS SQL Support will use and maintain the lengths of the EMPL_NAME and EMPL_NUMBER TEXT parameters received on the first execution, possibly resulting in TEXT parameter data truncation:

```
10120 ENTRY GET_NAME(EMPL_NAME,EMPL_NUMBER)
10130 .EXEC_SQL
10140 ..| FETCH C1 INTO :EMPL_NAME, :EMPL_NUMBER
10150 .END
10160 EXIT
```

To avoid the possibility of TEXT parameter data truncation, perform the following:

1. Move variables passed as ENTRY statement parameters to variables explicitly defined within the MANTIS program or subroutine.

2. Use the explicitly defined variables or host variables in the SQL statements, as shown below:

The following example shows host variables explicitly defined within the MANTIS subroutine. In this case, MANTIS SQL Support will always use TEXT variables of 30 and 12 characters, regardless of the size of the input parameters (EMPL_NAME, EMPL_NUMBER).

```
10110 ENTRY GET_NAME(EMPL_NAME,EMPL_NUMBER)
10120 .TEXT NAME(30),NUMBER(12):| define as maximal size
10130 .EXEC_SQL
10140 ..| FETCH C1 INTO :NAME, :NUMBER
10150 .END
10160 .EMPL_NAME=NAME
10170 .EMPL_NUMBER=NUMBER
10170 EXIT
```

4

Dynamic SQL statements

Using dynamic SQL statements in MANTIS SQL Support is somewhat different than using dynamic SQL statements in SQL in COBOL. If you are unfamiliar with dynamic SQL statements, you may want to review the appropriate SUPRA manuals for more information before reading this chapter.

Static and dynamic are the two basic types of SQL statements. You can use static SQL statements when you know all the information needed to execute the SQL statement before executing it. SQL SELECT, FETCH, UPDATE, INSERT, and DELETE are examples of static SQL statements. Use dynamic SQL statements when you do not know all the necessary information about an SQL statement before executing it. For example, if a query application allows a user to enter an SQL statement from an online terminal, the query program requires dynamic SQL statements. The SUPRA ISQL utility program is an example of such an application.

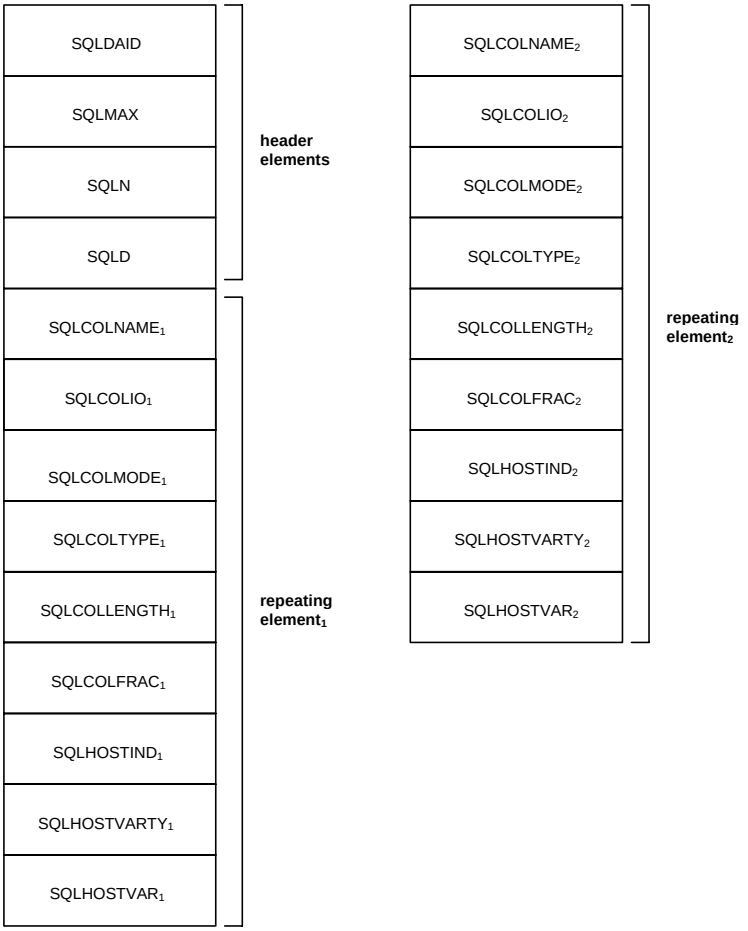
Dynamic SQL statements allow you to execute other SQL statements under control of the application program. PREPARE, DESCRIBE, EXECUTE, and EXECUTE IMMEDIATE are examples of dynamic SQL statements. Alternate forms of the DECLARE, OPEN, and FETCH statements can also be used with dynamic SQL statements.

You can use both static and dynamic SQL statements in the same MANTIS program. You cannot use host variables in SQL statements that are executed by dynamic SQL statements. A parameter marker (usually the question mark character (?)) must replace all host variables in the SQL statement text. Data is transferred between MANTIS and SUPRA using an SQLDA (SQL Descriptor Area). When using dynamic SQL statements, you must procedurally place host variable data into or retrieve data returned by SUPRA from an SQLDA.

To execute SQL statements using dynamic SQL, you must prepare the SQL statements with an SQL PREPARE statement and then execute them with an SQL EXECUTE statement. If data is retrieved, inserted, or updated in SUPRA, your program must manipulate an SQLDA between SQL statement preparation and execution. This manipulation can include allocating and expanding an SQLDA, retrieving data type and length information from SUPRA using the DESCRIBE statement, and transferring data between MANTIS variables and an SQLDA. Appendix A shows examples of MANTIS programs using static SQL statements and equivalent programs using dynamic SQL statements.

SQLDA statement and function

An SQLDA (SQL Descriptor Area) is a data structure that is used to transfer data between your program and the SUPRA when dynamic SQL statements are used. The following figure shows the structure of an SQLDA:



An SQLDA is composed of two types of elements: header elements (which occur once per SQLDA) and repeating elements (which can occur multiple times in an SQLDA). Repeating elements repeat as a group (each element occurs only once in a group). This repeating group is called an SQLVAR. Each element in the SQLDA has a specific name and contains a specific item of data. These items are explained in the SUPRA documentation.

In SQL in COBOL, you must explicitly declare each SQLDA element as a data area in your program and then access SQLDA elements through programming statements. In MANTIS SQL Support, you access the SQLDA by using the SQLDA statement and function. In MANTIS SQL Support, when you declare an SQLDA, an SQLDA with all the elements shown in the preceding figure is built for you. The SQLDA contains a default number of repeating elements (SQLVAR). Although your system administrator sets this default, you can modify this number in your program.

SQLDA is both a statement and a function. The SQLDA statement (write) stores data from the MANTIS program into the SQL Descriptor Area (SQLDA). The SQLDA function (read) transfers data from the SQLDA into the MANTIS program. For more detailed information, refer to *MANTIS Language, OS/390, VSE/ESA*, P39-5002.

The SQLDA statements are shown below followed by the SQLDA function. The SQLDA statement is used to allocate or deallocate an SQLDA, and to transfer data from a MANTIS program into an SQLDA.

Allocate an SQLDA

SQLDA(*sqlda_name*)=NEW

sqlda_name

Description *Required*. Specifies the name of the SQLDA to be allocated.

Format Must be a text literal or expression of 1–18 characters.

Consideration Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for SUPRA.

Example The following example shows how to allocate an SQLDA. Line 150 allocates an SQLDA named the “SQLDA1”.

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLMAX")<4
00170 .SQLDA("SQLDA1","SQLMAX")=4
00180 END
00190 SQLDA("SQLDA1","SQLN")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00230 END
00240 EXEC_SQL:| OPEN C1
00250 END
00260 EXEC_SQL:| PREPARE S2 FROM 'FETCH C1 USING DESCRIPTOR SQLDA1'
00270 END
00280 EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
00290 END
00300 EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
00310 END
00320 EMPL_NAME=SQLDA("SQLDA1","SQLHOSTVAR",1)
00330 EMPL_STREET=SQLDA("SQLDA1","SQLHOSTVAR",2)
00340 EMPL_STATE=SQLDA("SQLDA1","SQLHOSTVAR",3)
00350 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLHOSTVAR",4)
00360 SQLDA("SQLDA1")=QUIT
```

Deallocate an SQLDA SQLDA(sqlda_name)=QUIT

SQLDA(*sqlda_name*) = QUIT

sqlda_name

Description *Required*. Specifies the name of the SQLDA to be deallocated.

Format Must be a text literal or expression of 1–18 characters.

Considerations

- Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for SUPRA.
- Must be a previously defined SQLDA, via SQLDA(sqldaname) = NEW.

Example The following example shows how to deallocate an SQLDA. Line 360 deallocates an SQLDA named “SQLDA1”.

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE"
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLMAX")<4
00170 .SQLDA("SQLDA1","SQLMAX")=4
00180 END
00190 SQLDA("SQLDA1","SQLN")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00230 END
00240 EXEC_SQL:| OPEN C1
00250 END
00260 EXEC_SQL:| PREPARE S2 FROM 'FETCH C1 USING DESCRIPTOR SQLDA1'
00270 END
00280 EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
00290 END
00300 EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
00310 END
00320 EMPL_NAME=SQLDA("SQLDA1","SQLHOSTVAR",1)
00330 EMPL_STREET=SQLDA("SQLDA1","SQLHOSTVAR",2)
00340 EMPL_STATE=SQLDA("SQLDA1","SQLHOSTVAR",3)
00350 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLHOSTVAR",4)
00360 SQLDA("SQLDA1")=QUIT
```

Set SQLDA header information

SQLDA(*sqlda_name*,*sqlda_header_element*)=*expression*

sqlda_name

Description *Required*. Specifies the name of the SQLDA to be accessed.

Format Must be a text literal or expression of 1–18 characters.

Considerations

- Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for SUPRA.
- Must be a previously defined SQLDA, via SQLDA(sqldaname) = NEW.

sqlda_header_element

Description *Required*. Specifies the SQLDA header element that is accessed.

Format Must be a text literal or an expression that evaluates to one of the SQLDA header element names shown in the table following the next parameter description.

expression

Description *Required*. Specifies the data to be transferred from the MANTIS program into the SQLDA.

Format Must be consistent with the datatype of the SQLDA element being stored (either text or numeric).

Consideration Certain SQLDA elements are read-only and cannot have data from the MANTIS program stored in them. In some cases, storing data in one SQLDA element causes MANTIS to automatically update other SQLDA elements.

Example In the following example, SQLDA header elements are set in lines 250 and 270.

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 EMPL_NAME="JONES"
00140 EMPL_STREET=" NEW STREET ADDRESS"
00150 EMPL_STATE="OH"
00160 EMPL_ZIP_CODE="12345"
00170 SQL_TEXT="UPDATE EMPLOYEE.TABLE SET"
00180 SQL_TEXT=SQL_TEXT+" NAME = ?, STREET = ?,"
00190 SQL_TEXT=SQL_TEXT+" STATE = ?, ZIP_CODE = ?"
00200 SQL_TEXT=SQL_TEXT+" WHERE NAME = ?"
00210 EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
00220 END
00230 SQLDA ("SQLDA1")=NEW
00240 IF SQLDA("SQLDA1","SQLMAX")<5
00250 .SQLDA("SQLDA1","SQLMAX")=5
00260 END
00270 SQLDA("SQLDA1","SQLN")=5
00280 SQLDA("SQLDA1","SQLHOSTVAR",1)=EMPL_NAME
00290 SQLDA("SQLDA1","SQLHOSTVAR",2)=EMPL_STREET
00300 SQLDA("SQLDA1","SQLHOSTVAR",3)=EMPL_STATE
00310 SQLDA("SQLDA1","SQLHOSTVAR",4)=EMPL_ZIP_CODE
00320 SQLDA("SQLDA1","SQLHOSTVAR",5)=EMPL_NAME
00330 EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
00340 END
00350 SQLDA("SQLDA1")=QUIT
```

The following table lists and describes the SQLDA header elements:

Element name	MANTIS compatible data type	How set / when used	Updateable?
SQLDAID	TEXT(8)	Eyecatcher. Set by SUPRA.	No
SQLMAX	BIG	Number of repeating groups (SQLVAR) in the SQLDA. Set using installation defined default value when the SQLDA is allocated. Can be modified by the MANTIS program if needed.	Yes
SQLN	BIG	Total number of repeating groups in use in the SQLDA. Set as a result of a DESCRIBE to the total number of host variable parameters in the statement (except for DESCRIBE in FETCH USING DESCRIPTOR where SQLN is set to the number of result table columns).	Yes
SQLD	BIG	Total number of output host variables in the SQLDA. Set as a result of a DESCRIBE to the number of host variables (except for DESCRIBE in FETCH USING DESCRIPTOR where SQLD is set to the number of result table columns).	Yes

Move data into an SQLDA repeating group

SQLDA(*sqlda_name*,*repeating_element*,*index*)=*expression*

sqlda_name

Description *Required*. Specifies the name of the SQLDA to be accessed.

Format Must be a text literal or expression of 1–18 characters.

Consideration Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for SUPRA.

sqlda_repeating_element

Description *Required*. Specifies the repeating element of the SQLDA that is accessed.

Format Must be a text literal or expression that evaluates to one of the SQLDA repeating element names shown in the table at the end of this section.

index

Description *Required* when accessing repeating elements. Specifies the group of SQLDA repeating elements that is accessed.

Format Must be a numeric literal or expression not less than one and not greater than the maximum number of repeating groups currently in the SQLDA (SQLMAX), inclusive.

General consideration

Using certain SQLDA elements causes other SQLDA elements to automatically be set by MANTIS SQL Support. For example, storing data into the SQLDA element “SQLHOSTVAR” causes MANTIS to set the datatype (“SQLHOSTVARTY”) and data length (“SQLCOLLEN”) elements automatically.

Example In the following example, lines 280–320 store data from the MANTIS program into SQLDA repeating group elements.

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 EMPL_NAME="JONES"
00140 EMPL_STREET=" NEW STREET ADDRESS"
00150 EMPL_STATE="OH"
00160 EMPL_ZIP_CODE="12345"
00170 SQL_TEXT="UPDATE EMPLOYEE.TABLE SET"
00180 SQL_TEXT=SQL_TEXT+" NAME = ?, STREET = ?,"
00190 SQL_TEXT=SQL_TEXT+" STATE = ?, ZIP_CODE = ?"
00200 SQL_TEXT=SQL_TEXT+" WHERE NAME = ?"
00210 EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
00220 END
00230 SQLDA ("SQLDA1")=NEW
00240 IF SQLDA("SQLDA1","SQLMAX")<5
00250 .SQLDA("SQLDA1","SQLMAX")=5
00260 END
00270 SQLDA("SQLDA1","SQLN")=5
00280 SQLDA("SQLDA1","SQLHOSTVAR",1)=EMPL_NAME
00290 SQLDA("SQLDA1","SQLHOSTVAR",2)=EMPL_STREET
00300 SQLDA("SQLDA1","SQLHOSTVAR",3)=EMPL_STATE
00310 SQLDA("SQLDA1","SQLHOSTVAR",4)=EMPL_ZIP_CODE
00320 SQLDA("SQLDA1","SQLHOSTVAR",5)=EMPL_NAME
00330 EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
00340 END
00350 SQLDA("SQLDA1")=QUIT
```

Element name	MANTIS compatible data type	Contents / considerations	Updateable?
SQLCOLNAME	TEXT(18)	SQL column name. Set by SUPRA as the result of a DESCRIBE; can be set by the MANTIS program.	Yes
SQLCOLIO	BIG	Indicates whether host variable is input or output. Set by SUPRA as the result of a DESCRIBE.	No
SQLCOLMODE	BIG	Indicates whether NULL values are allowed. Set as a result of a DESCRIBE.	No
SQLCOLTYPE	BIG	Data type as it resides on the database. Set by SUPRA as a result of a DESCRIBE.	No
SQLCOLLENGTH	BIG	Total number of bytes used to store the data. Set by SUPRA as the result of a DESCRIBE, or by MANTIS when data is transferred by SQLHOSTVAR.	No
SQLCOLFRAC	BIG	Number of decimal positions for FIXED column types. Set by SUPRA as the result of a DESCRIBE. Not used by MANTIS because all numeric data is floating point.	No
SQLHOSTIND	BIG	Contains the value of the indicator variable. Used to indicate the presence of NULL variables and truncated data. Set by SUPRA during SQL function; can be set by the MANTIS program.	Yes
SQLHOSTVARTY	BIG	Contains the datatype of the data in the SQLDA. Set by MANTIS when SQLHOSTVAR is used.	No
SQLHOSTVAR	TEXT BIG KANJI	Subfunction that physically transfers data between MANTIS data areas and the SQLDA data areas. Used to transfer value of variable between SUPRA and MANTIS.	Yes

Read header elements

SQLDA(*sqlda_name*,*sqlda_header_element*)

sqlda_name

Description *Required*. Specifies the name of the SQLDA to be accessed.

Format Must be a text literal or expression of 1–18 characters.

Consideration Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for SUPRA.

sqlda_header_element

Description *Required*. Specifies the SQLDA header element that is accessed.

Format Must be a text literal or an expression that evaluates to one of the SQLDA header element names shown in the table at the end of the section titled “Set SQLDA header information” which begins on page 51.

Example In the following example, line 160 shows the SQLDA “SQLMAX” header element being read from the SQLDA.

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLMAX")<4
00170 .SQLDA("SQLDA1","SQLMAX")=4
00180 END
00190 SQLDA("SQLDA1","SQLN")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00230 END
00240 EXEC_SQL:| OPEN C1
00250 END
00260 EXEC_SQL:| PREPARE S2 FROM 'FETCH C1 USING DESCRIPTOR SQLDA1'
00270 END
00280 EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
00290 END
00300 EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
00310 END
00320 EMPL_NAME=SQLDA("SQLDA1","SQLHOSTVAR",1)
00330 EMPL_STREET=SQLDA("SQLDA1","SQLHOSTVAR",2)
00340 EMPL_STATE=SQLDA("SQLDA1","SQLHOSTVAR",3)
00350 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLHOSTVAR",4)
00360 SQLDA("SQLDA1")=QUIT
```

Move data from an SQLDA repeating group into a MANTIS program

SQLDA(*sqlda_name*,*sqlda_repeating_element*,*index*)

sqlda_name

Description *Required*. Specifies the name of the SQLDA to be accessed.

Format Must be a text literal or expression of 1–18 characters.

Consideration Naming conventions for SQL entities must be followed. Refer to the appropriate SQL language manual for SUPRA.

sqlda_repeating_element

Description *Required*. Specifies the repeating element of the SQLDA that is accessed.

Format Must be a text literal or expression that evaluates to one of the SQLDA repeating element names. See the table at the end of the section titled "Move data into an SQLDA repeating group" which begins on page 54.

index

Description *Required* when accessing repeating elements. Specifies the group of SQLDA repeating elements that is accessed.

Format Must be a numeric literal or expression between one and the maximum number of repeating groups currently in the SQLDA (SQLMAX), inclusive.

Example In the following example, lines 320–350 show SQLDA repeating elements being moved from the SQLDA into the MANTIS program.

```
00100 TEXT SQL_TEXT(254)
00110 TEXT EMPL_NAME(30),EMPL_STREET(30),EMPL_STATE(2)
00120 BIG EMPL_ZIP_CODE
00130 SQL_TEXT="SELECT NAME, STREET, STATE, ZIPCODE"
00140 SQL_TEXT=SQL_TEXT+" FROM EMPLOYEE.TABLE
00150 SQLDA("SQLDA1")=NEW
00160 IF SQLDA("SQLDA1","SQLMAX")<4
00170 .SQLDA("SQLDA1","SQLMAX")=4
00180 END
00190 SQLDA("SQLDA1","SQLN")=4
00200 EXEC_SQL:| PREPARE S1 FROM: SQL_TEXT
00210 END
00220 EXEC_SQL:| DECLARE C1 CURSOR FOR S1
00230 END
00240 EXEC_SQL:| OPEN C1
00250 END
00260 EXEC_SQL:| PREPARE S2 FROM 'FETCH C1 USING DESCRIPTOR SQLDA1'
00270 END
00280 EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
00290 END
00300 EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
00310 END
00320 EMPL_NAME=SQLDA("SQLDA1","SQLHOSTVAR",1)
00330 EMPL_STREET=SQLDA("SQLDA1","SQLHOSTVAR",2)
00340 EMPL_STATE=SQLDA("SQLDA1","SQLHOSTVAR",3)
00350 EMPL_ZIP_CODE=SQLDA("SQLDA1","SQLHOSTVAR",4)
00360 SQLDA("SQLDA1")=QUIT
```

SQL statements larger than 254 characters

To dynamically prepare an SQL statement for execution, use the SQL PREPARE statement. The SQL statement to be executed is contained either in the PREPARE statement as a text literal, or in the host variable specified in the PREPARE statement.

Because the SQL statement is text, the host variable specified in the PREPARE statement must be a MANTIS TEXT variable. MANTIS TEXT variables are limited to 254 characters, which can cause a problem for large SQL statements.

MANTIS SQL Support allows large SQL statements to be placed in a MANTIS TEXT array. MANTIS SQL Support combines the contents of all rows in a MANTIS TEXT array into a single SQL statement. This combined text is then used as the SQL statement text in the SQL PREPARE statement.

In order to use this feature, the host variable specified in the PREPARE statement:

- Must be declared as a MANTIS TEXT array.
- Must not have subscripts specified. If subscripts are present, MANTIS uses only the text of the subscripted row as the SQL statement.

When these conditions are met, MANTIS uses the contents of the entire array as the SQL statement text. The text is combined “as is” from all rows in the array. No characters are added or deleted so that SQL text may split row boundaries if required. Rows containing no data are ignored.

Because ALL rows of the array are combined, some rows may need to be cleared from one SQL statement execution to another. If they are not cleared, an SQL statement may be combined with text remaining from a previous SQL statement. Executing this statement may cause errors that are difficult to diagnose.

An example of how to use a MANTIS TEXT array to contain a large SQL statement is shown below:

```
40 ..TEXT SQL_TEXT(10,20)
```

Array can be any length/number.

```
 50 ..SQL_TEXT(1)="SELECT EMPLNO, EMPLN"
 60 ..SQL_TEXT(2)="M"
 70 ..SQL_TEXT(5)="FROM EMPLOYEE.TABLE"
 80 ..SQL_TEXT(6)="WHERE EMPLNO= ?"
```

Text can split array boundaries.

Rows 3 & 4 ignored-contains no data.

No spaces are added or deleted.

```
90 ..EXEC_SQL
100 ...| PREPARE S1 FROM :SQL_TEXT
110 ..END
```

No subscripts specified.

The SQL text used by the SQL PREPARE statement in this example is:

```
SELECT EMPLNO, EMPLNM FROM EMPLOYEE.TABLE WHERE EMPLNO= ?
```

A

Sample MANTIS SQL support programs

This appendix contains the following sample MANTIS SQL Support programs:

- Insert program using static SQL statements
- Insert program using dynamic SQL statements
- Update program using static SQL statements
- Update program using dynamic SQL statements
- Select program using static SQL statements
- Select program using dynamic SQL statements
- Delete program using static SQL statements
- Delete program using dynamic SQL statements
- Column select program using dynamic SQL statements

Each example program using dynamic SQL statements is equivalent to the corresponding program using static SQL statements. For clarity, the examples do not contain error checking or display logic, and the information to be transferred to the database is coded in the programs as literals.

Insert program using static SQL statements

This program inserts one employee into an employee table using static SQL statements.

```
 10 ENTRY STATIC_INSERT
 20 .BIG HIRE_DATE,BIRTH_DATE,JOB_CODE,SALARY,EDUCATION_LEVEL
 30 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20),MIDDLE_INITIAL(1),LAST_NAME(20)
.40 .TEXT PHONE_NUMBER(4),WORK_DEPARTMENT(3),SEX(1)
 50 .|
 60 .EMPLOYEE_NUMBER="000120"
 70 .FIRST_NAME="SEAN"
 80 .MIDDLE_INITIAL=" "
 90 .LAST_NAME="O'CONNELL"
100 .BIRTH_DATE=421018
110 .HIRE_DATE=631205
120 .JOB_CODE=58
130 .EDUCATION_LEVEL=14
140 .SALARY=29250
150 .PHONE_NUMBER="2167"
160 .WORK_DEPARTMENT="A00"
170 .SEX="M"
180 .|
190 .EXEC_SQL:| INSERT INTO  DSN82.TEMPL
200 ..|                      (EMPNO,
210 ..|                      FIRSTNME,
220 ..|                      MIDINIT,
230 ..|                      LASTNAME,
240 ..|                      BRTHDATE,
250 ..|                      HIREDATE,
260 ..|                      JOBCODE,
270 ..|                      EDUCLVL,
280 ..|                      SALARY,
290 ..|                      PHONENO,
300 ..|                      WORKDEPT,
310 ..|                      SEX)
320 ..|              VALUES (:EMPLOYEE_NUMBER,
330 ..|                      :FIRST_NAME,
340 ..|                      :MIDDLE_INITIAL,
350 ..|                      :LAST_NAME,
360 ..|                      :BIRTH_DATE,
370 ..|                      :HIRE_DATE,
380 ..|                      :JOB_CODE,
390 ..|                      :EDUCATION_LEVEL,
400 ..|                      :SALARY,
410 ..|                      :PHONE_NUMBER,
420 ..|                      :WORK_DEPARTMENT,
430 ..|                      :SEX)
440 .END
450 EXIT
```

Insert program using dynamic SQL statements

This program inserts one employee into an employee table using dynamic SQL statements.

```
 10 ENTRY DYNAMIC_INSERT
 20 .BIG HIRE_DATE,BIRTH_DATE,JOB_CODE,SALARY,EDUCATION_LEVEL
 30 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20),MIDDLE_INITIAL(1),LAST_NAME(20)
 40 .TEXT PHONE_NUMBER(4),WORK_DEPARTMENT(3),SEX(1)
 50 .TEXT SQL_TEXT(254)
 60 .|
 70 .EMPLOYEE_NUMBER="000120"
 80 .FIRST_NAME="SEAN"
 90 .MIDDLE_INITIAL=" "
100 .LAST_NAME="O'CONNELL"
110 .BIRTH_DATE=421018
120 .HIRE_DATE=631205
130 .JOB_CODE=58
140 .EDUCATION_LEVEL=14
150 .SALARY=29250
160 .PHONE_NUMBER="2167"
170 .WORK_DEPARTMENT="A00"
180 .SEX="M"
190 .|
200 .SQL_TEXT="INSERT INTO DSN82.TEMPL "
210 .'"(EMPNO, FIRSTNME, MIDINIT, LASTNAME, BRTHDATE, "
220 .'"HIREDATE, JOBCODE, EDUCLVL, SALARY, PHONENO, "
230 .'"WORKDEPT, SEX) "
240 .'"VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
250 .|
260 .EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
270 .END
280 .|
290 .SQLDA("SQLDA1")=NEW
300 .SQLDA("SQLDA1","SQLMAX")=12
310 .SQLDA("SQLDA1","SQLN")=12
320 .SQLDA("SQLDA1","SQLHOSTVAR",1)=EMPLOYEE_NUMBER
330 .SQLDA("SQLDA1","SQLHOSTVAR",2)=FIRST_NAME
340 .SQLDA("SQLDA1","SQLHOSTVAR",3)=MIDDLE_INITIAL
350 .SQLDA("SQLDA1","SQLHOSTVAR",4)=LAST_NAME
360 .SQLDA("SQLDA1","SQLHOSTVAR",5)=BIRTH_DATE
370 .SQLDA("SQLDA1","SQLHOSTVAR",6)=HIRE_DATE
380 .SQLDA("SQLDA1","SQLHOSTVAR",7)=JOB_CODE
390 .SQLDA("SQLDA1","SQLHOSTVAR",8)=EDUCATION_LEVEL
400 .SQLDA("SQLDA1","SQLHOSTVAR",9)=SALARY
410 .SQLDA("SQLDA1","SQLHOSTVAR",10)=PHONE_NUMBER
420 .SQLDA("SQLDA1","SQLHOSTVAR",11)=WORK_DEPARTMENT
430 .SQLDA("SQLDA1","SQLHOSTVAR",12)=SEX
440 .|
450 .EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
460 .END
470 EXIT
```

Update program using static SQL statements

This program updates one employee in an employee table using static SQL statements.

```
 10 ENTRY STATIC_UPDATE
 20 .BIG HIRE_DATE,BIRTH_DATE
 30 .TEXT EMPLOYEE_NUMBER(6)
 40 .TEXT FIRST_NAME(20),MIDDLE_INITIAL(1),LAST_NAME(20)
 50 .|
 60 .EMPLOYEE_NUMBER="000120"
 70 .FIRST_NAME="JOHN"
 80 .MIDDLE_INITIAL="H"
 90 .LAST_NAME="DOE"
100 .BIRTH_DATE=490113
110 .HIRE_DATE=880120
120 .|
130 .EXEC_SQL
140 ..|
150 ..| UPDATE  DSN82.TEMPL
160 ..|
170 ..|   SET FIRSTNME = :FIRST_NAME,
180 ..|       MIDINIT  = :MIDDLE_INITIAL,
190 ..|       LASTNAME = :LAST_NAME,
200 ..|       BRTHDATE = :BIRTH_DATE,
210 ..|       HIREDATE = :HIRE_DATE
220 ..|
230 ..|   WHERE EMPNO = :EMPLOYEE_NUMBER
240 .END
250 EXIT
```

Update program using dynamic SQL statements

This program updates one employee in an employee table using dynamic SQL statements.

```
 10 ENTRY DYNAMIC_UPDATE
 20 .BIG HIRE_DATE,BIRTH_DATE
 30 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20),MIDDLE_INITIAL(1),LAST_NAME(20)
 40 .TEXT DA(18),DAPARM(8)
 50 .TEXT SQL_TEXT(254)
 60 .|
 70 .EMPLOYEE_NUMBER="000120"
 80 .FIRST_NAME="JOHN"
 90 .MIDDLE_INITIAL="H"
100 .LAST_NAME="DOE"
110 .BIRTH_DATE=490113
120 .HIRE_DATE=880120
130 .|
140 .SQL_TEXT="UPDATE DSN82.TEMPL SET "
150 .SQL_TEXT=SQL_TEXT+"FIRSTNME = ?, MIDINIT = ?, LASTNAME = ?, "
160 .SQL_TEXT=SQL_TEXT+"BRTHDATE = ?, HIREDATE = ? "
170 .SQL_TEXT=SQL_TEXT+"WHERE EMPNO = ? "
180 .|
190 .EXEC_SQL:| PREPARE S1 FROM :SQL_TEXT
200 .END
210 .|
220 .SQLDA("SQLDA1")=NEW
230 .DA="SQLDA1"
240 .DAPARM="SQLHOSTVAR"
250 .SQLDA(DA,"SQLMAX")=6
260 .SQLDA(DA,"SQLN")=6
270 .SQLDA(DA,DAPARM,1)=FIRST_NAME
280 .SQLDA(DA,DAPARM,2)=MIDDLE_INITIAL
290 .SQLDA(DA,DAPARM,3)=LAST_NAME
300 .SQLDA(DA,DAPARM,4)=BIRTH_DATE
310 .SQLDA(DA,DAPARM,5)=HIRE_DATE
320 .SQLDA(DA,DAPARM,6)=EMPLOYEE_NUMBER
330 .|
340 .EXEC_SQL:| EXECUTE S1 USING DESCRIPTOR SQLDA1
350 .END
360 EXIT
```

Select program using static SQL statements

This program retrieves employee information for one employee from an employee table using static SQL statements.

```
10 ENTRY STATIC_SELECT
20 .BIG HIRE_DATE,BIRTH_DATE,JOB_CODE,SALARY,EDUCATION_LEVEL
30 .TEXT EMPLOYEE_NUMBER(6),FIRST_NAME(20),MIDDLE_INITIAL(1),LAST_NAME(20)
40 .TEXT WORK_DEPARTMENT(3),PHONE_NUMBER(3),SEX(1)
50 .EMPLOYEE_NUMBER="000120"
60 .|
70 .EXEC_SQL:| DECLARE C1 CURSOR FOR
80 ..|            SELECT * FROM DSN82.TEMPL
90 ..|              WHERE EMPNO = :EMPLOYEE_NUMBER
100 .END
110 .EXEC_SQL:| OPEN C1
120 .END
130 .EXEC_SQL:| FETCH C1 INTO  :EMPLOYEE_NUMBER,
140 ..|                        :FIRST_NAME,
150 ..|                        :MIDDLE_INITIAL,
160 ..|                        :LAST_NAME,
170 ..|                        :BIRTH_DATE,
180 ..|                        :HIRE_DATE,
190 ..|                        :JOB_CODE,
200 ..|                        :EDUCATION_LEVEL,
210 ..|                        :SALARY,
220 ..|                        :PHONE_NUMBER,
230 ..|                        :WORK_DEPARTMENT,
240 ..|                        :SEX
250 .END
260 .EXEC_SQL:| CLOSE C1
270 .END
280 EXIT
```

Select program using dynamic SQL statements

This program retrieves employee information for one employee from an employee table using dynamic SQL statements.

```
 10 ENTRY DYNAMIC_SELECT
 20 .BIG HIRE_DATE,BIRTH_DATE,JOB_CODE,SALARY,EDUCATION_LEVEL
 30 .TEXT EMPLOYEE_NUMBER(6)
 40 .TEXT FIRST_NAME(20),MIDDLE_INITIAL(1),LAST_NAME(20)
 50 .TEXT WORK_DEPARTMENT(3),PHONE_NUMBER(3),SEX(1)
 60 .TEXT SQL_TEXT(254)
 70 .|
 80 .EMPLOYEE_NUMBER="000120"
 90 .SQL_TEXT="SELECT * FROM DSN82.TEMPL "
100 .'"WHERE EMPNO=? "
110 .|
120 .EXEC_SQL:|  PREPARE S1 FROM :SQL_TEXT
130 .END
140 .EXEC_SQL:|  DECLARE C1 CURSOR FOR S1
150 .END
160 .EXEC_SQL:|  OPEN C1 USING :EMPLOYEE_NUMBER
170 .END
180 .SQL_TEXT="FETCH C1 USING DESCRIPTOR"
190 .EXEC_SQL:| PREPARE S2 FROM :SQL_TEXT
200 .END
210 .SQLDA("SQLDA1")=NEW
220 .EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
230 .END
240 .EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
250 .END
260 .EXEC_SQL:|  CLOSE C1
270 .END
280 .|
290 .EMPLOYEE_NUMBER=SQLDA("SQLDA1","SQLHOSTVAR",1)
300 .FIRST_NAME=SQLDA("SQLDA1","SQLHOSTVAR",2)
310 .MIDDLE_INITIAL=SQLDA("SQLDA1","SQLHOSTVAR",3)
320 .LAST_NAME=SQLDA("SQLDA1","SQLHOSTVAR",4)
330 .BIRTH_DATE=SQLDA("SQLDA1","SQLHOSTVAR",5)
340 .HIRE_DATE=SQLDA("SQLDA1","SQLHOSTVAR",6)
350 .JOB_CODE=SQLDA("SQLDA1","SQLHOSTVAR",7)
360 .EDUCATION_LEVEL=SQLDA("SQLDA1","SQLHOSTVAR",8)
370 .SALARY=SQLDA("SQLDA1","SQLHOSTVAR",9)
380 .PHONE_NUMBER=SQLDA("SQLDA1","SQLHOSTVAR",10)
390 .WORK_DEPARTMENT=SQLDA("SQLDA1","SQLHOSTVAR",11)
400 .SEX=SQLDA("SQLDA1","SQLHOSTVAR",12)
410 EXIT
```

Delete program using static SQL statements

This program deletes one employee from an employee table using static SQL statements.

```
 10 ENTRY STATIC_DELETE
 20 .TEXT EMPLOYEE_NUMBER(6)
 30 .EMPLOYEE_NUMBER "000120"
 40 .EXEC_SQL
 50 ..|
 60 ..|   DELETE FROM DSN82.TEMPL
 70 ..|
 80 ..|       WHERE EMPNO = :EMPLOYEE_NUMBER
 90 .END
100 EXIT
```

Delete program using dynamic SQL statements

This program deletes one employee from an employee table using dynamic SQL statements.

```
10 ENTRY DYNAMIC_DELETE
20 .TEXT EMPLOYEE NUMBER(6),SQL TEXT(254)
30 .EMPLOYEE_NUMBER "000120"
40 .SQL_TEXT="DELETE FROM DSN82.TEMPL WHERE EMPNO = ? "
50 .|
60 .EXEC_SQL
70 ..|
80 ..|   PREPARE S1 FROM :SQL_TEXT
90 ..|
100 .END
110 .EXEC_SQL
120 ..|
130 ..|   EXECUTE S1 USING :EMPLOYEE_NUMBER
140 ..|
150 .END
160 EXIT
```

Column select program using dynamic SQL statements

This program uses dynamic SQL statements to retrieve column names, data types, lengths, and the first row of the columns from a table specified by the user.

```
 10 ENTRY SQL_LIST_TABLES
 20 .|
 30 .| THIS PROGRAM LISTS COLUMNS BASED ON TABLE NAME
 40 .|
 50 .TEXT TABLE_NAME(32)
 60 .TEXT SQL_FUNCTION(100)
 70 .SHOW"PLEASE ENTER TABLE NAME "
 80 .OBTAIN TABLE_NAME
 90 .SQL_FUNCTION="SELECT * FROM" +TABLE_NAME
100 .EXEC_SQL
110 ..| PREPARE S1 FROM :SQL_FUNCTION
120 .END
130 .EXEC_SQL:| DECLARE C1 CURSOR FOR S1
140 .END
150 .EXEC_SQL:| OPEN C1
160 .END
170 .SQL_FUNCTION="FETCH C1 USING DESCRIPTOR"
180 .EXEC_SQL:| PREPARE S2 FROM :SQL_FUNCTION
190 .END
200 .SQLDA("SQLDA1")=NEW
210 .EXEC_SQL:| DESCRIBE S2 INTO SQLDA1
220 .END
230 .EXEC_SQL:| EXECUTE S2 USING DESCRIPTOR SQLDA1
240 .END
250 .EXEC_SQL:| CLOSE C1
260 .END
280 .SHOW"COLUMN NAME",AT(45),"LENGTH",AT(55),"DATA"
290 .FOR COUNTER=1 TO SQLDA("SQLDA1","SQLN")
300 ..SHOW SQLDA("SQLDA1","SQLCOLNAME",COUNTER),
310 ..'AT(25),TXT(SQLDA("SQLDA1","SQLCOLTYPE",COUNTER)),
320 ..'AT(45),TXT(SQLDA("SQLDA1","SQLCOLLENGTH",COUNTER)),
330 ..'AT(55),(SQLDA("SQLDA1","SQLHOSTVAR",COUNTER))
350 .END
360 .WAIT
370 EXIT
```

B

Features not supported

The following features are not supported:

- The USING LABELS clause of the DESCRIBE statement is not implemented.
- Host variables may not be specified in a SELECT list. In the example below, the VX host variable is invalid:

```
SELECT A,:VX,C
INTO :VA,:VB,:VC
```

- Exact line number reference when syntax errors are detected is not supported in all cases. Once control is transferred to SUPRA to execute an SQL statement, MANTIS no longer has control and therefore does not know on which line the error occurred. For example, if an error occurred in the INTO clause of the following statement:

```
01330 ..X=X+1
01340 ..EXEC_SQL
01350 ...|SELECT A,B,C
01360 ...|INTO :VA,:VB),:VC      <--- Error in this line
01370 ...|FROM TABLE.1
01380 ...|WHERE A=1
01390 ..END
01400 ..X=X-VA
```

MANTIS will point to the beginning of the SQL statement as being in error. For example:

```
01330 ..X=X+1
01340 ..EXEC_SQL
===>0 ...|SELECT A,B,C          <--- MANTIS points to this line
01360 ...|INTO:VA,:VB),:VC      <--- Error in this line
01370 ...|FROM TABLE.1
01380 ...|WHERE A=1
01390 ..END
01400 ..X=X-VA
```

- The contents of one SQLDA structure cannot be implicitly copied into another in a single instruction. The following statement is not permitted:

```
SQLDA("NAME2")=SQLDA("NAME1")
```

However, each element of an SQLDA can be moved individually to the corresponding element of a different SQLDA.

C

Comparing SQL in MANTIS SQL support to SQL in COBOL

The following general considerations apply to SQL in MANTIS SQL Support as compared to SQL in COBOL:

- SQL statements are embedded in a MANTIS application program as MANTIS comments. Each SQL statement is bracketed with an EXEC_SQL-END block. No MANTIS comments are permitted within the EXEC_SQL-END block. All comments within the block are considered SQL statement text.
- In the SQL WHENEVER statement:
  - A MANTIS DO statement replaces the GOTO clause, and FAULT replaces STOP.
  - The default for the SQLERROR condition is FAULT; in SQL in COBOL, the default is CONTINUE.
  - WHENEVER settings may have different ranges of applicability than they would in SQL in COBOL.
- SQLCA elements are accessed through the SQLCA function and statement, rather than as items of data.
- SQLDA elements are accessed through the SQLDA function and statement, rather than as items of data.
- MANTIS SQL Support does not support the SQL INCLUDE statement. The SQLCA and SQLDA functions eliminate the need to INCLUDE these structures.
- In MANTIS, quotation marks (") delimit character-string constants. In SQL, apostrophes (‘) delimit character-string constants.
- SQL data types are supported in MANTIS compatible data types. These are listed in the following table.

Permissible data type conversions between SQL and MANTIS are listed in the following table. Note that this table is an exact replica of the table under "Converting data between MANTIS SQL support and SUPRA" on page 26; it is repeated here for convenience.

SQL data type	MANTIS data type	Notes
0 (Fixed)	BIG/SMALL	Loss of precision may occur when converting from SQL to MANTIS. Overflow may occur when converting from MANTIS to SQL.
1 (Float)	BIG/SMALL	Loss of precision may occur when converting from SQL to MANTIS. Overflow may occur when converting from MANTIS to SQL.
2 (Character)	TEXT	When converting in either direction, truncation may occur.
3 (Byte)	TEXT	When converting in either direction, truncation may occur.
4 (Date)	TEXT	When converting in either direction, truncation may occur.
5 (Time)	TEXT	When converting in either direction, truncation may occur.
6 (String)	TEXT	When converting in either direction, truncation may occur.
10 (DBYTE)	KANJI	When converting in either direction, truncation may occur.

- When dynamic SQL statements are used, only data type codes for MANTIS-compatible data types are returned in the SQLHOSTVARTY element in the SQLDA. Valid data types are thus limited to those listed in the preceding table.

- When a null value is encountered in an optional column and the user did not code a host indicator variable for the same column, MANTIS and COBOL yield different results at run-time.

  Using SUPRA/SQL with MANTIS:

  - An SQLCODE of 0 is returned by SUPRA to the user program. This is because MANTIS internally has host variable indicators for each host variable.

  - Depending on how the C$OPCUST option SQLNDTA is specified:

    If SQLNDTA=N, the user host variable is not updated if the null indicator was internally received for the column.

    If SQLNDTA=Y, the user host variable is cleared if the null indicator was internally received for the column.

  For SUPRA/SQL with COBOL using static SQL statements, the precompiled program result at run-time is:

  - An SQLCODE of -809 (error) is returned by SUPRA to the user program. Refer to the *Supra Messages and Codes Reference Manual (SQL),* P26-0128*,* message code CSWP0809/-0809, for more information on this message.

  - COBOL stops moving data to user host variables once the null indicator is received.

D

SQL keywords

This appendix lists the SQL keywords used by MANTIS SQL Support. SQL keywords are not MANTIS reserved words and can be used in MANTIS SQL programs. However, if SQL keywords are used as host variable names in SQL statements, errors can occur. MANTIS SQL Support does limited parsing of the SQL statement text before passing the SQL statement to SUPRA for execution. During parsing, host variables, which are the same as SQL keywords, may be taken as keywords instead of host variables causing a MANTIS fault. In the example below, MANTIS could display a fault message because ":COLUMN" is used as a host variable but is an SQL keyword.

```
.TEXT COLUMN(10), COLUMN1(20), COLUMN2(10
.EXEC_SQL
..| SELECT COLUMN, COLUMN1, COLUMN2
..|   INTO :COLUMN, :COLUMN1,:COLUMN2
..|   FROM TABLEA
.END
```

The following table lists the SQL keywords used by MANTIS SQL Support.

ALL	AS	
BEGIN	BUFFER	BY
CLOSE	CONNECT	CURRENT
COLUMN	CONTINUE	CURSOR
COMMIT	COPY	
DATE	DELETE	DO
DBNAME	DESCRIBE	DROP
DECLARE	DESCRIPTOR	

END EXECUTE

FAULT FIRSTPOS FOUND
FETCH FOR FROM
FIRST

HOLD

IDENTIFIED INDEX INTO
IMMEDIATE INDEXNAME IS
INCLUDE INSERT

KEY

LAST LENGTH

NEXT NOT

OF OPEN OUT

PACKAGESET PREPARE PROGRAM
POS PREV PUT

READ RESET ROLLBACK
RELEASE

SAME SERVER SQLID
SEARCH SET SQLWARNING
SECTION SQLERROR STATISTICS
SELECT SQLEXCEPTION

TABLE TIMESTAMP TO
TIME TIMEZONE

UNION USER USING
UPDATE

VALUES

WHENEVER WITH WORK
WHERE

Index

R

S

T

V

W